Towards a Software Transactional Memory for heterogeneous CPU-GPU processors

Alejandro VILLEGAS [1], Angeles NAVARRO, Rafael ASENJO and Oscar PLATA
[a] *Universidad de Málaga, Andalucía Tech.*
Dept. Computer Architecture, 29071 Málaga, Spain
email: {avillegas, angeles, asenjo, oscar}@ac.uma.es

Abstract. The heterogeneous Accelerated Processing Units (APUs) integrate a multi-core CPU and a GPU within the same chip. Modern APUs provide the programmer with platform atomics, used to communicate the CPU cores with the GPU using simple atomic datatypes. However, ensuring consistency for complex data types is a task delegated to programmers, who have to implement a mutual exclusion mechanism. Transactional Memory (TM) is an optimistic approach to implement mutual exclusion. With TM, shared data can be accessed by multiple computing threads speculatively, but changes are only visible if a transaction ends with no conflict with others in its memory accesses. TM has been studied and implemented in software and hardware for both CPU and GPU platforms, but an integrated solution has not been provided for APU processors.

In this paper we present APUTM, a software TM designed to work on heterogeneous APU processors. The design of APUTM focuses on minimizing the access to shared metadata in order to reduce the communication overhead via expensive platform atomics. The main objective of APUTM is to help us understand the tradeoffs of implementing a sofware TM on an heterogeneous CPU-GPU platform and to identify the key aspects to be considered in each device. In our experiments, we compare the adaptability of APUTM to execute in one of the devices (CPU or GPU) or in both of them simultaneously. These experiments show that APUTM is able to outperform sequential execution of the applications.

Keywords. transactional memory, APU processors, parallel programming

1. Introduction

Nowadays, multi-threaded programming has become essential in order to take the most of multi-core CPUs, GPUs, and many other kinds of processors. One of the main issues programmers face when designing their algorithms is the management of shared data: if two or more computing threads share a piece of data, then programmers have to implement a mechanism that ensures mutual exclusion. Usually, mutual exclusion is implemented employing locking mechanism that are inefficient (if coarse-grained locks are implemented) or hard to program (if the solution uses fine-grained locks).

[1] Corresponding Author

Transactional Memory (TM) [9] has emerged as a promising alternative to the use of locks. The TM interface allows programmers to enclose the accesses to shared data inside *transactions*, which execute in parallel in a speculative way. Mutual exclusion is guaranteed thanks to the implementation of proper version management and conflict detection mechanisms. As TM is proving to both ease programming and to obtain substantial speedups in CPUs [8] and GPUs [1,11,10,6,5,2] separately, the industry keeps moving to integrate both devices into a single heterogeneous processor. The Accelerated Processing Units (APUs) are available for some years now. These processors incorporate a multi-core CPU and a GPU in the same chip. Not only that, but the Heterogeneous Systems Architecture (HSA)[2] initiative is setting standards to ease programming and define the expected capabilities of heterogeneous platforms. Specifically, HSA-enabled APUs offer an unified memory space visible by both CPU and GPU, and a set of platform-atomics operations that allow for communication between both devices. As programmers start to deploy their programs across the CPU cores and the GPU, the needs of an efficient mutual exclusion mechanism will surface and exploring different implementations of TM solutions will be an important research field.

The implementation of TM in heterogeneous CPU-GPU processors is a challenging task, as both CPU and GPU work under different programming models. Multi-core CPUs follow the MIMD model, where multiple cores may operate on different data or a shared memory, while GPUs follow the SIMD programming model, where multiple threads execute the same instruction (lockstep execution) on different memory positions. Another challenge is the memory space. CPUs have access to a main memory via a coherent cache hierarchy. On the GPU, main memory is accessed via a cache hierarchy where, in most cases, the L1 data cache is not coherent. In addition, GPUs feature a low-latency scratch-pad memory (shared memory in CUDA, local memory in OpenCL, or tiled memory in C++AMP) that can be used to accelerate the management of GPU-private transactional metadata. Lastly, communication between CPU and GPU is an important problem to solve. Platforms atomics ensure that values communicate effectively between both devices, but these operations are expensive in terms of memory latency.

To better understand and overcome these challenges, we propose APUTM, a software TM designed to work on APU processors. APUTM can be configured to run separately on the CPU cores or the GPU, or simultaneously in both devices, ensuring mutual exclusion in any case. APUTM is inspired by NOrec [3], combining a fast timestamp-based conflict detection mechanism with a precise value-based validation. For the GPU, APUTM implements a mechanism to allow for parallel commits of transactions while updating the timestamp information to communicate with the CPU cores. In our evaluation, we use a synthetic workload and two microbenchmarks to analyze different configurations of APUTM. We provide a discussion on the impact that our design decisions have on the performance of APUTM on both devices and provide hints for future improvements.

2. Conclusions and Future work

In this paper we present APUTM, a software TM designed specifically for APU architectures. Two implementations are presented: one using a global sequence lock to com-

[2] http://www.hsafoundation.com/

mit transactions quickly, and the other checking for conflicts using a transaction-private read-set. In general, the former presents better performance in both the CPU and the GPU, but is more scalable when running on the CPU. The main goal of APUTM is to understand the behavior of transactions on APU processors. Besides that, APUTM is able to outperform sequential execution in some scenarios.

We plan to improve APUTM in several ways: coalescing GPU memory accesses for a better use of memory bandwidth, creating a scheduler for a better distribution of transactions across devices, and increasing the granularity of GPU commits from wavefront to work-group to increase performance. We also consider proposing architectural changes to reduce the overhead of enabling/disabling GPU transactions that conflict. Lastly, we plan to extend the evaluation of APUTM. We are currently implementing more applications to assess APUTM, as well as studying the overhead of the library with respect to the total execution time and other TM-related metrics.

Acknowledgements

This work has been supported by projects TIN2013-42253-P and TIN2016-80920-R, from the Spanish Government, P11-TIC8144 and P12- TIC1470, from Junta de Andalucía, and Universidad de Málaga, Campus de Excelencia Internacional, Andalucía Tech.

References

[1] D. Cederman, P. Tsigas, and M. T. Chaudhry. Towards a software transactional memory for graphics processors. In *10th Eurographics Conf. on Parallel Graphics and Visualization (EG PGV'10)*, pages 121–129, 2010.

[2] S. Chen and L. Peng. Efficient GPU hardware transactional memory through early conflict resolution. In *22nd Int'l. Symp. on High Performance Computer Architecture (HPCA'16)*, 2016.

[3] L. Dalessandro, M. F. Spear, and M. L. Scott. Norec: Streamlining stm by abolishing ownership records. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 67–78, New York, NY, USA, 2010. ACM.

[4] P. Felber, C. Fetzer, T. Riegel, and P. Marlier. Time-based software transactional memory. *IEEE Transactions on Parallel & Distributed Systems*, 21:1793–1807, 2010.

[5] W. W. L. Fung and T. M. Aamodt. Energy efficient GPU transactional memory via space-time optimizations. In *46th Ann. IEEE/ACM Int'l. Symp. on Microarchitecture (MICRO'13)*, pages 408–420, 2013.

[6] W. W. L. Fung, I. Singh, A. Brownsword, and T. M. Aamodt. Hardware transactional memory for GPU architectures. In *44th Ann. IEEE/ACM Int'l. Symp. on Microarchitecture (MICRO'11)*, pages 296–307, 2011.

[7] R. Guerraoui and M. Kapalka. On the correctness of transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '08, pages 175–184, New York, NY, USA, 2008. ACM.

[8] T. Harris, J. Larus, and R. Rajwar. *Transactional Memory, 2nd Ed.* Morgan & Claypool Publishers, USA, 2010.

[9] M. Herlihy and J. E. B. Moss. Transactional Memory: Architectural support for lock-free data structures. In *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, pages 289–300, 1993.

[10] A. Holey and A. Zhai. Lightweight software transactions on GPUs. In *43rd Int'l Conf. on Parallel Processing (ICPP'14)*, pages 461–470, 2014.

[11] Y. Xu, R. Wang, N. Goswami, T. Li, L. Gao, and D. Qian. Software transactional memory for GPU architectures. In *Ann. IEEE/ACM Int'l. Symp. on Code Generation and Optimization (CGO'14)*, pages 1:1–1:10, 2014.